Intro. C-strings Sec. 4.5

Pointers to structures.

))) (with Profs. 1, 2)

Copying structures with pointers:

CSI 310: Lecture 7 (No lab assignment this week. Use lab time for TA help)
pointer variable to hold the address returned by new. 

Reminder: A Dynamic Variable is useless without a (separate)

<table>
<thead>
<tr>
<th>Dynamic Variable</th>
<th>Automatic Structure</th>
<th>Structure/Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ . . . new bag2(SZ) ; } ;</td>
<td>{ . . . bag2 MyBag(SZ) ; } ;</td>
<td></td>
</tr>
<tr>
<td>int (size = t SZ) ;</td>
<td>int (size = t SZ) ;</td>
<td></td>
</tr>
<tr>
<td>#include &quot;bag2.h&quot;</td>
<td>#include &quot;bag2.h&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Array</th>
<th>Automatic Array</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ . . . new int [SZ] ; } ;</td>
<td>{ . . . int x[98] ; } ;</td>
<td></td>
</tr>
<tr>
<td>int (size = t SZ) ;</td>
<td>int (size = t SZ) ;</td>
<td></td>
</tr>
<tr>
<td>#include &quot;bag2.h&quot;</td>
<td>#include &quot;bag2.h&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Integer</th>
<th>Automatic Integer</th>
<th>Scalar (Integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ . . . new int ; } ;</td>
<td>{ . . . int x ; } ;</td>
<td></td>
</tr>
<tr>
<td>int (size = t SZ) ;</td>
<td>int (size = t SZ) ;</td>
<td></td>
</tr>
<tr>
<td>#include &quot;bag2.h&quot;</td>
<td>#include &quot;bag2.h&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Variable</th>
<th>Automatic Variable</th>
<th>More Orthogonality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
structures. That is the subject of Chapter 5 and beyond.

Unlike the auto. variables, dynamic vars. have no

names.

PBag is a pointer to a structure. (Structures can contain pointers to

AutoBag. data is a pointer to an array.

Unlike the auto. variables,

Automatic variables

belonging to this call to main

''head'' or ''free store'' of dynamic variables

// includes of improved bag

#include "bag2.h" // Ch. 4 improved bag

#include "bag2.h" // Ch. 4 improved bag

#include "bag2.h" // Ch. 4 improved bag

#include "bag2.h" // Ch. 4 improved bag
2. Illegal Pointers.

RECYCLED memory?

1. DANGLING Pointers: Pointers to Garbage or

Two (2) BIG Pitfalls of Pointers:

will create a DANGLING POINTER (value in page).

delete page.

Now,
This is a "CRASH": Computer tried to read memory at the illegal address 0x0.

Segmentation fault

cout >> *MP >> end;

0x0 0x0e0f04

cout >> *MP >> " " " >> *MP >> " " end;

cout >> *MP >> end;

After Assignment:

4006

MYP

NULL

Before Assignment:

4006

MYP

MYP = NULL;
Or, is it the value of a pointer variable, that is, an address?

So, when you or others say "pointer," think hard: Is it a pointer variable?

But most everyone, we and DSO, say for short, "PIVAR is a pointer."

Illegal value: 

C/++ int variable, or else it might have the NULL value, or else some pointer type variable. The variable named PIVAR might store an address of a

We and DSO said "a pointer is an address. PIVAR is really (the name of)

int *PIVAR; What is PIVAR? ? ? Is it a "pointer" ? ? ?

int PIVAR; What is PIVAR? ? ? It's an integer.

int IVAR; What is IVAR? ? ?

A linguistic pitfall—try not to fall into it!
Perhaps we should always use the word "address" for "pointer value".

done.

type which determines what values it can hold and what operations can be performed on them. Each variable has a type. 

Technically, "pointer" and "int" describe C/++ types. Each variable has a
A single pointer locates the array. The fastest way to control sequential processing of the array from beginning to end.

If you can live without the terminating value, 

\[ A \]

It's the advantages of way 2.

**EXCEEDING**

For both ways, the allocated space must not be

\[ 183-186 \frac{7}{1} \]

Example: C-string pages

next entity containing a terminating value. Example:

2. Data structure = the array ONLY. The end of used prefix is marked by the

1. Data structure = array (variable) + "used" count variable.

Where does the used data prefix end? Two ways to tell:

array is "idle".

The used data is stored in a contiguous prefix of the array. The rest of the

Partially filled array: dynamically allocated or not.
Advantages of way I.

(C) Item values are not restricted: No special terminating value is required.

(B) Appending a new item to the end is fast (also constant time).

(A) You can tell the number of items right away "in constant time"
char type

"Null-Terminated Strings" or "C-strings"
1. A variable of type `char` can hold a character of the implementation character set.
   ```
   char ch = 'A';  // 'A' is a character literal.
   ```

2. Sizes of C/C++, variables are expressed as multiples of the size of a `char`:
   So, by definition, `sizeof(char) = 1`.

3. A `char` has at least 8 bits, guaranteed. 8-bit `chars` are almost universal.

4. `chars` are automatically converted to and from `ints`. The $256 = 2^8$
   different 8-bit `chars` convert to 0 to 255, or to $-128$ to $127$, depending on
   `YOUR C/C++ implementation`.

5. The most popular characters, about which most the World agrees, are the
   7-bit ASCII character set given in Appendix A of DSO. This ASCII table give the correspondance

   what printers print $\longleftrightarrow$ `int` conversion value

6. All data in a program's process is stored and computed in (current)
E is 69

....

cout " " " Is " >> I >> I end;

int I = ch; //char to int conversion.
char ch = 'E';

...  

using namespace std; //So cout means std::cout

#include <iostream>

# include <iostream>

Why letter E?

These same manufactures!

of the printer

Social convention

Why 69?

Base 2

Why 69?

numeral

system:

I+4+64

128

64

32

16

8

4

2

1

Base 2

Base 2

8 bits

These same

computers in terms of binary digits or bits.

make it add

unit

adding arithmetic

sent to an ASCII printer

make it print sent to an

University at Albany Computer Science Dept.
Unix and other system interface libraries use C-strings.

#include <cstring> // Library has very useful functions.
#include <iostream> // Facilities "know about" C-strings.
0 = |s ->
...

```cpp
{  count >> ch >> II |s = "-|s";
  char I = ch;
  char ch = "\0\n",;
} main()

using namespace std;
```
A C-string is a partially-filled array of char variables whose terminating element is the null character.
```cpp
{ 
    count >> boost::sizeReports >> sizeReport(ACSTR) >> end;
    count >> "last char in ACSTR has int value:" >> last >> end;
    int last = ACSTR[coutem];
    count >> coutem >> end;
    count >> "length of ACSTR = " >> ACSTR.length >> end;
    count >> 
    coutem++;
    else
        break; // terminate the while loop
if (ACSTR[coutem] == '0')
    while (true)
        int coutem = 0;
    char ACSTR = [] { I'm a string of 29 characters";
main()
using namespace std;
#include <iostream>

University at Albany Computer Science Dept.
16
BUT size=30
Last char in AcSr has int value: 0
Length of "I'm a string of 29 characters" = 29
29 ordinary characters plus the 1 null terminating character!

assert(sizeof(ACstr)==30)

char ACstr[]="I'm a string of 29 characters";